

## Programming Models for High-Performance Computing

Rusty Lusk

Mathematics and Computer Science Division

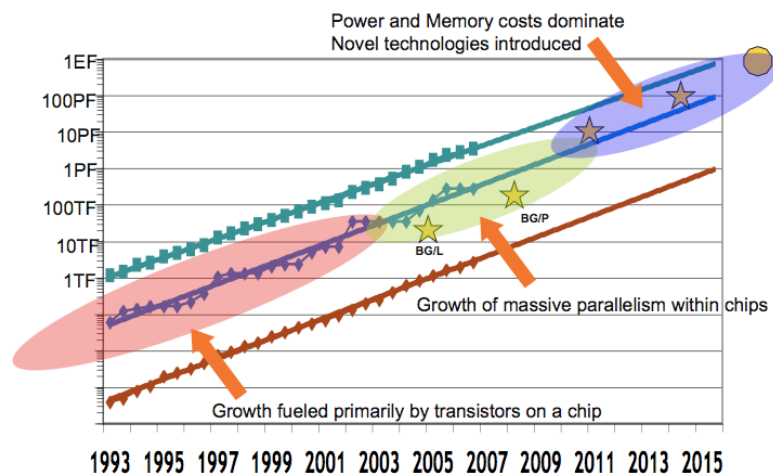
Argonne National Laboratory

## Outline of the Situation

- Million core systems and beyond are on the horizon
- Today labs and universities have general purpose systems with 10k-200K cores (BGL@ LLNL 200K, BGP@Argonne 160K, XT5@ORNL 150K cores)
- By 2012 there will be more systems deployed in the 200K-1M core range
- By 2020 there will be systems with perhaps 100M cores
- Personal systems with > 1000 cores within 5
- Personal systems with requirement for 1M threads is not too far fetched (GPUs for example)



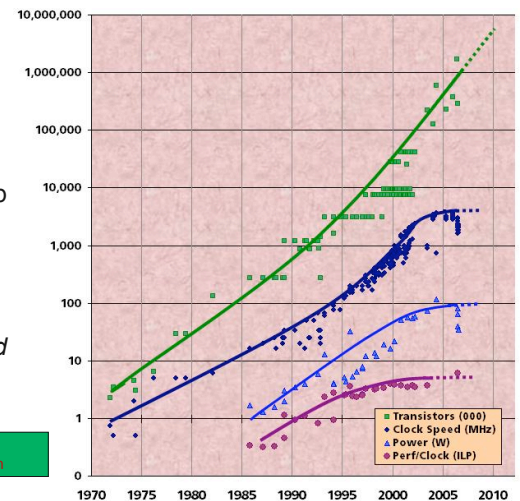
## Looking out to Exascale... Concurrency will be Doubling every 18 months

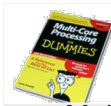


## Traditional Sources of Performance Improvement are Flat-Lining (2004)

- New Constraints
  - 15 years of *exponential* clock rate growth has ended
- Moore's Law reinterpreted:
  - How do we use all of those transistors to keep performance increasing at historical rates?
  - Industry Response: #cores per chip doubles every 18 months *instead* of clock frequency!

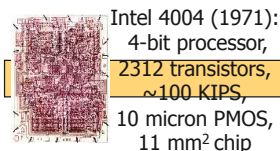
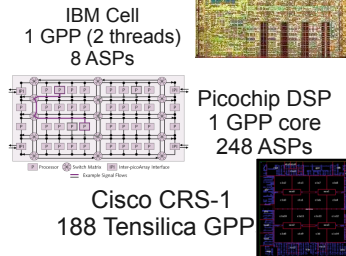
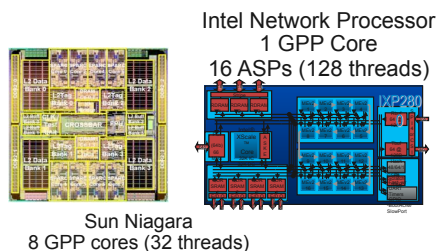
Figure courtesy of Kunle Olukotun, Lance Hammond, Herb Sutter, and Burton Smith





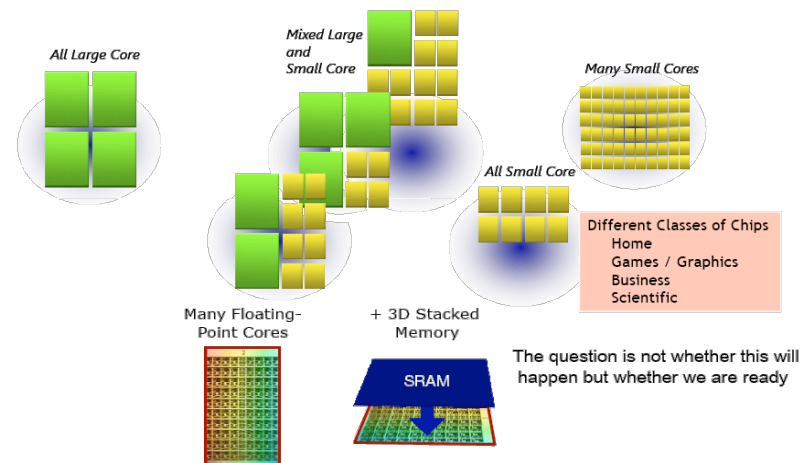
## Multicore comes in a wide variety

- Multiple parallel general-purpose processors (GPPs)
- Multiple application-specific processors (ASPs)



*"The Processor is the  
new  
Transistor" [Rowen]*

## What's Next?



Source: Jack Dongarra, ISC 2008

and this is just for the individual cores



## How Will We Program Them?

- Still an unsolved problem
  - Many approaches described at SC'09
    - especially for GPUs
- Some believe a totally new programming model and language will be needed.
- Some mechanism for dealing with shared memory will be necessary
  - This (whatever it is) plus MPI is the conservative view
- Whatever it is, it will need to interact properly with MPI
- May also need to deal with on-node heterogeneity
- The situation is somewhat like message-passing before MPI
  - And it is too early to standardize



## MPI is Current HPC Programming Model

- MPI represents a very complete definition of a well-defined programming model
- MPI programs are *portable*
- There are many implementations
  - Vendors
  - Open source
- Enables high performance for wide class of architectures
  - Scalable algorithms are key
- Small subset easy to learn and use
- Expert MPI programmers needed most for libraries, which are encouraged by the MPI design.



## The MPI Forum Continues to Refresh MPI

- New signatures for old functions
  - E.g. MPI\_Send(...,MPI\_Count,...)
- Details
  - Fortran binding issues..
- New features
  - MPI\_Process\_Group and related functions for fault tolerance
  - New topology routines aware of more hierarchy levels
  - Non-blocking collective operations
  - A simpler one-sided communication interface
    - Or perhaps standardized semantics for interacting with shared-memory programming systems in general
  - More scalable versions of the “v” collectives
  - ...
- See <http://www.mpi-forum.org> for details of working groups

## Why Won't “MPI Everywhere” suffice?

- Core count on a node is increasing faster than memory size.
- Thus memory available per MPI process is going down.
- Thus we need parallelism *within* an address space, while continuing to use MPI for parallelism among separate address spaces.
- We don't have a good way to do this yet.
- Whatever we use, it must cooperate with parallelism across address spaces, so its API must interact in a well-defined way with MPI.
- Some applications are expressing the need for large address spaces that span multiple multi-core nodes, yet still are each a small part of the memory of the entire machine.

## Moving Beyond MPI

- Any alternative to MPI (at its own level) will have to have some of the good properties of MPI
  - Portability
  - Scalability
  - Performance
- Perhaps alternatives exist at different levels.
- But they will still have to interact with MPI, in order to provide a path from where we are now to more abstract models
  - Clear interoperability semantics
  - Can be used either above or below C/Fortran/MPI code

## Some Families of Programming Models and Associated Languages

- Shared-memory and annotation languages
  - Especially OpenMP
  - Likely to coexist with MPI
- Partitioned Global Address Space Languages
  - UPC, Co-Array Fortran, and Titanium
  - One step removed from MPI
- The HPCS languages
  - X10, Chapel, Fortress
  - Two steps removed from MPI

## OpenMP

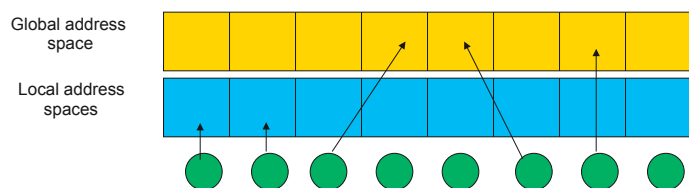
- OpenMP is a set of compiler directives (in comments, like HPF) plus some library calls
- The comments direct the execution of loops in parallel in a convenient way.
- Data placement is not controlled, so performance is hard to get except on machines with real shared memory.
- Likely to be more successful on multicore chips than on previous SMP's (multicore = really, *really* shared memory).
- Can co-exist with MPI
  - MPI's levels of thread safety correspond to programming constructs in OpenMP
    - *Formal methods can be applied to hybrid programs*
- New book by Barbara Chapman, et al.

## Other Annotation-based approaches

- The idea is to retain the sequential programming model
- Annotations guide source-to-source transformations or compilation into a parallel program
- HPF and OpenMP (part 1) are examples
- Others in research mode

## The PGAS Languages

- PGAS (Partitioned Global Address Space) languages attempt to combine the convenience of the global view of data with awareness of data locality, for performance
  - Co-Array Fortran, an extension to Fortran-90
  - UPC (Unified Parallel C), an extension to C
  - Titanium, a parallel version of Java



- Fixed number of processes, like MPI-1

## PGAS Languages Status

- Compilers exist
  - In some cases more than one
- Applications are being tried
- Substantial support, at least for UPC
- Early experiments are encouraging with respect to performance
  - Some reports are misleading.
- Little take-up by scientific applications so far
- PGAS BOF at SC'09

## The DARPA HPCS Language Project

- The DARPA High Productivity Computer Systems (HPCS) Project is a 10-year, three-phase, hardware/software effort to transform the productivity aspect of the HPC enterprise.
- In Phase II, three vendors were funded to develop high productivity language systems, and each assigned a small group to language development
  - IBM: X10
  - Cray: Chapel
  - Sun: Fortress
- In Phase III, Sun was dropped from DARPA support. Both IBM and Cray efforts are continuing. Actually, Sun's effort is too, internally supported.
- Two steps removed from MPI: not a fixed number of processes

## Quasi Mainstream Programming Models

- C, Fortran, C++ and MPI
- OpenMP, pthreads
- (CUDA, RapidMind, Cn) → OpenCL
- PGAS (UPC, CAF, Titanium)
- HPCS Languages (Chapel, Fortress, X10)
- HPC Research Languages and Runtime
- HLL (Parallel Matlab, Grid Mathematica, etc.)

## Hybrid Programming Models

- Some shared-memory API's that can be used with MPI
  - POSIX threads -- explicit thread creation, locks, condition vars
  - OpenMP
    - Sequential programming model with annotations, parallel execution model
  - Yet to be invented...
- The current situation: OpenMP + MPI
  - Works because of well-thought-out explicit contracts between the models.
    - MPI standard defines levels of thread safety
    - OpenMP defines types of code regions
    - These work together in ways defined by the respective standards
  - Hard to get performance with OpenMP because of lack of locality management, excessive synchronization.

## One Possible Near Future: PGAS+MPI

- Locality management within an address space via local, remote memory
- An address space could be bigger than one node
  - Might need more hierarchy in PGAS definitions
- Just starting to work with PGAS folks on UPC+MPI and CAF+MPI
  - Center for Programming Models base program project with ANL, LBNL, Rice, Houston, PNNL, OSU
- Until recently PGAS has focused either on competing with MPI or with OpenMP on single node
  - Need to make interoperability with MPI a priority to attract current HPC applications

## A More Distant Future

- HPCS-type languages have many interesting ideas for exploiting less obvious parallelism
- Need coordination and freedom from vendor ownership
- A convergence plan
  - (DARPA briefly funded a convergence project, which was promising until cancelled)
- A migration plan for current applications
  - Interaction with MPI
  - Use in libraries
- Both Chapel and X10 highly visible in HPC Challenge at SC '09
  - Benchmarks, not full applications

## Libraries

- Libraries are an easier way to implement programming models than languages
  - need old linker, not new compiler
- Libraries can hide complexity of MPI (or other programming model instantiation)
- Libraries can provide special-purpose programming models
  - still with applicability across applications
- Library implementation would be the next step in applying new programming approaches like PGAS or HPCS languages
  - will need to work with existing programming environment, other compilers and languages
  - This would provide a migration path for applications
- My current work is on the ADLB (Asynchronous, Dynamic Load-Balancing) library
  - scalable implementation of the master/slave programming model

## The Transition is Starting

- In large-scale scientific computing today essentially all codes are message-passing based. Additionally many are starting to use some form of multithreading on SMP or multicore nodes.
- Multicore is challenging programming models but there has not yet emerged a dominate model to augment message passing
- There is a need to identify new hierarchical programming models that will be stable over long term and can support the concurrency doubling pressure
- Current approaches to programming GPU's are for library developers, not application developers

## Summary

- MPI is a successful current standard, but emerging architectures will force us to look at new approaches
- Most immediate need: a shared-memory programming model that interacts well with MPI
- Next need, and approach to programming heterogeneous multi-core processors that is suitable for HPC computers and application scientists
- Programming models for exascale are still in experimental stages
- Hiding MPI calls in higher-level libraries can be a useful approach to programmer productivity

The End